P05 Treasure Hunt Adventure Game

Overview

This assignment involves developing a treasure hunt style adventure game. This game will include graphical elements that the user can click on, and can drag onto others to find clues, solve puzzles and eventually win the game. The provided graphics and level file contain a treasure hunt game set in a backyard. This can be an amazing adventure game for children. The child must find a coin and place/drop it onto a target location showed in the map to win. A screenshot of what the screen of this game may look like after you have completed this assignment is shown in Fig. 1. A demo of this application is provided in this video.

Figure 1: A screenshot of the treasure hunt game

Learning Objectives

The goals of this assignment include:

- Experience organizing code in an object oriented fashion that takes advantage of inheritance relationships between your classes.
- Use of inheritance and interfaces to better organize your code in a more clear and concise manner. Students should also enjoy the power of polymorphism.
- Learn how to use `PApplet` class defined in the processing graphic library directly to develop a graphic application from the scratch, rather than through a provide .jar wrapper file.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P05 pre-assignment quiz is accessible through Canvas before having access to this specification by **11:59PM on Sunday 03/20/2022**.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
20 points	**Additional Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but may test different parts of your submission in different ways.
5 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission with respect to the commenting and style requirements defined in the CS300 Course Style Guide. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.

Assignment Requirements and Notes

- Pair programming is **ALLOWED** for this assignment. If you decide to work with a partner on this assignment, REGISTER your partnership NO LATER than **11:59PM on Sunday 03/20/2022** and MAKE SURE that you have read and understood the CS300 Pair Programming Policy. Your partner must be enrolled in one of the cs300 spring 2022 lectures (L1, L2, L3, or L4).
- Make sure to read carefully through the specification provided in this write-up. This assignment requires clear understanding of its instructions. Read TWICE the instructions

and do not hesitate to ask for clarification on piazza if you find any ambiguity.

- The ONLY import statements that you may include in your classes are:

```
import processing.core.PApplet;
import processing.core.PImage;
import java.util.ArrayList;
import java.util.NoSuchElementException;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
```

- The automated grading tests in gradescope are **NOT using the full processing library** when grading your code. They only know about the following fields and methods (referenced directly from this assignment). If you are using any other fields, methods, or classes from the processing library, this will cause problems for the automated grading tests. Such references must be replaced with references to one or more of the following:
  - **two** fields from `PImage`: int `width`, and int `height`.
  - **six** fields from `PApplet`: int `mouseX`, int `mouseY`, boolean `mousePressed`, int `width`, int `height`, and boolean `focused`.
  - the following methods from `PApplet`: `loadImage()`, `image()`, `imageMode()`, `getSurface()`, `save()`, `textAllign()`, `textSize()`, `text()`, `stroke()`, `fill()`, `rectMode()`, `rect()`, and `main()`.

  If you are using any other fields, methods, or classes from the `PImage` class, this will cause problems for the automated grading tests.

- You CAN define local variables that you may need to implement the methods defined in this program.

- You CAN define private methods to help implement the different public methods defined in this write-up, if needed.

- You MUST NOT add any additional fields either instance or static to your program, and any **public** methods either static or instance to your program, other than those defined in this write-up.

- Any source code provided in this specification may be included verbatim in your program without attribution.

- **NONE** of the callback methods (`setup(), draw(), mousePressed(), and mouseReleased()`) defined in the `TreasureHunt` class should be called explicitly in this program.

- **Callback Methods**: A callback method specifies additional computation that should happen in response to specific events. In this assignment, we are going to use the following callback methods.
  - `setup()` method: This method is automatically called when the program begins. It creates and initializes the different data fields defined in your program, and configures the different graphical settings of your application, such as loading the background image, setting the dimensions of the display window, etc.
  - `draw()` method: This method continuously executes the lines of code contained inside its block until the program is stopped. It draws repeatedly the display window to the screen. All processing programs update the screen at the end of the draw() method.
  - `mousePressed()`: The block of code defined in this method is automatically executed each time the mouse bottom is pressed.
  - `mouseReleased()`: The block of code defined in this method is automatically executed each time the mouse bottom is released.
- All the String comparisons in this assignment are CASE-SENSITIVE.
- You can submit your work in progress multiple times on gradescope. Your submission may include methods not implemented or with partial implementation or with a default return statement. But avoid submitting a code which does not compile. A submission which contains compile errors won't pass any of the automated tests on gradescope.
- Your submitted source files must include at the top a complete file header.
- All implemented methods including the overridden methods MUST have their own javadoc-style method headers, with accordance to the CS300 Course Style Guide.
- An `@Override` annotation must be added at the top of every overridden method.
- All your classes MUST have a javadoc-style class header.
- You MUST adhere to the Academic Conduct Expectations and Advice

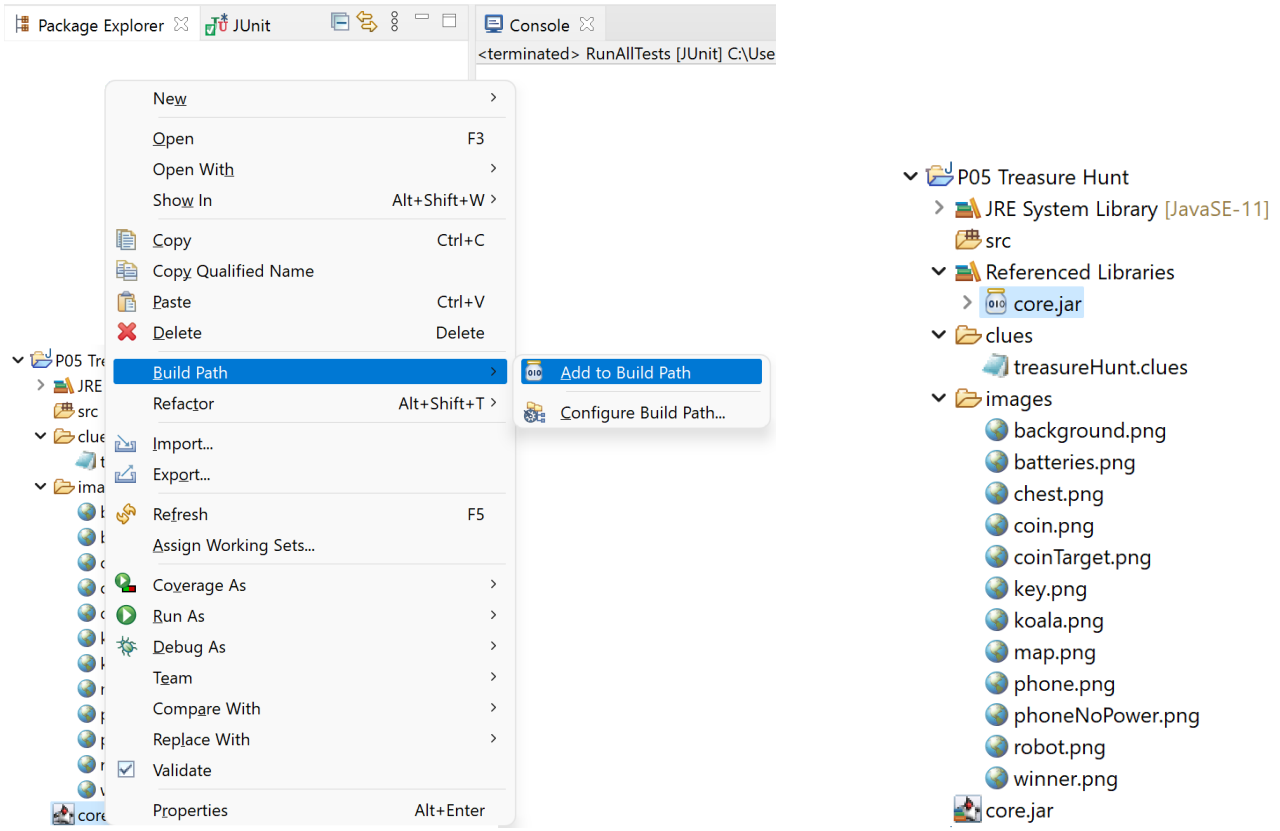
1 Getting Started

1. To get started, let's first create a new Java11 project within Eclipse. You can name this project whatever you like, but "P05 Treasure Hunt" is a descriptive choice. You have to ensure that your new project uses Java 11, by setting the "Use an execution environment JRE:" drop down setting to "JavaSE-11" within the new Java Project dialog box. Do not create a module associated to your java project. All your source java files must be included within the default src package of your p05 project.
2. Next, download the following file p05Distributables.zip. Then, extract the contents of the archive file directly into your P5 project folder. This should add a core.jar file, a folder of images, and a folder of clues to your project folder.

3. The **core.jar** file is part of the processing graphical library that you are welcome (but not required) to read more about here. If this .jar file does not immediately appear in the Project Explorer, try right-clicking your project in the project folder and selecting "Refresh" to fix that. To make use of the code within this jar file, you'll need to right-click on it within the Project Explorer and choose **"Add to Build Path"** from the "Build Path" menu.

4. The **images** folder contains **12** png image files and the **clues** folder contains only single file named `treasureHunt.clues`.

5. Figure 2 illustrates how your project will look like in eclipse package explorer after completing this step.



(a) Adding core.jar to build path (b) P05 package explorer

Figure 2: Treasure Hunt Application - Getting Started

2 Define the Clickable interface

You may have noticed while watching the demo of p05 Treasure Hunt application that most of the objects in the backyard are **clickable**. A clickable object responds to the mouse being pressed (while it is over the object) and released. All these objects are repeatedly drawn to the display window in different ways, and respond differently to the mouse events (when the mouse is over them, pressed, or released). So let's first define the **interface** `Clickable`.

Create a new `interface` called `Clickable` and add it to your p05 project. This interface should include the following four abstract methods.

```
public void draw(); // draws this Clickable object to the screen
public void mousePressed(); // defines the behavior of this Clickable object
                            // each time the mouse is pressed
public void mouseReleased(); // defines the behavior of this Clickable object
                            // each time the mouse is released
public boolean isMouseOver(); // returns true if the mouse is over this clickable object
```

3 Overview of the code organization using inheritance and interfaces

In this graphic application, we are going to consider the following clickable objects:

- **Interactive objects** which hold clues and/or trigger other interactive objects to appear in the backyard. All the interactive objects are clickable. Some of them are draggable (such as the key, the batteries, the coin). Other clickable objects are not draggable (such as the chest, the robot, the map, the phone). Draggable objects can be droppable leading to some action when they are dropped into their target. We are going to organize these interactive objects into a hierarchy of classes.
- **Buttons**: this application defines two buttons. A button to restart the game and a button to take a screenshot of the game scene. The color of the button changes from a dark to light blue if the mouse is over it (being selected) or not.

Figure 3 illustrates the class diagram of this treasure hunt application. This diagram models at a high level of abstraction the different classes that you are going to implement in this assignment and their relationship. It also includes the interface `Clickable`. This hierarchy of classes is documented in these javadocs. You can explore these javadocs but do not create any class yet. We are going to create and implement these classes in the next steps.

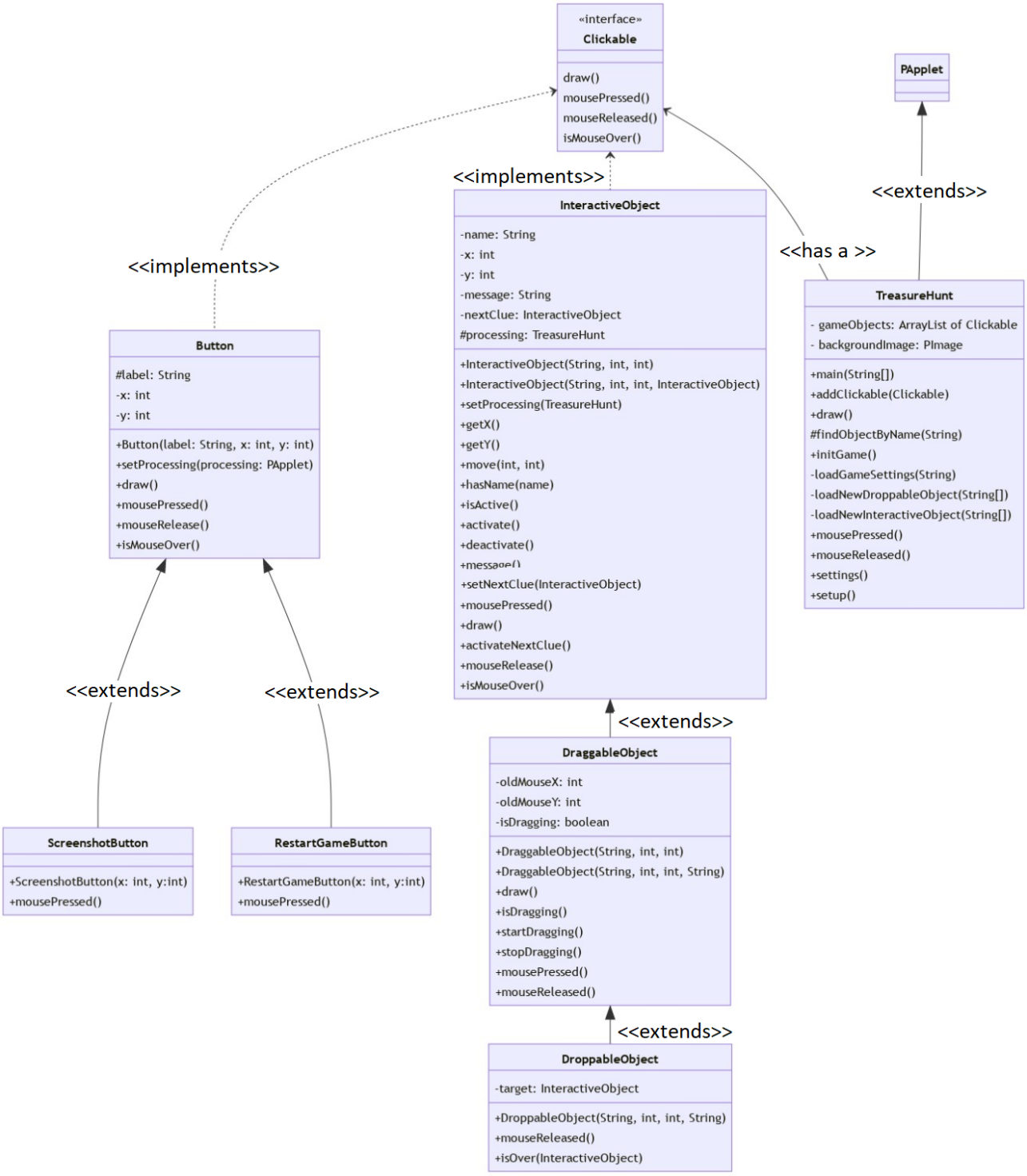


Figure 3: P05 classes and their relationships

You can notice that we are going to consider the following classes.

- `TreasureHunt extends PApplet` defining an **is-a** inheritance relationship. The TreasureHunt class is of type PApplet. This means that it models the display window of our graphic adventure game.
- The class TreasureHunt **has a** list of objects instances of the `Clickable` interface.
- All the clickable objects (including interactive objects or buttons) must share the same display window instance of PApplet.
- `InteractiveObject implements Clickable` defining also an **is-a** relationship. Every interactive object MUST implement all the methods defined in the `Clickable` interface.
- `DraggableObject extends InteractiveObject`. Every draggable object is clickable but also can start dragging and follows the mouse moves when clicked.
- `DroppableObject extends DraggableObject`. Every droppable object is draggable. This means that it is also a clickable interactive object. While being dragged, a droppable object can be dropped on another interactive object resulting on some action to be performed.
- `Button implements Clickable`. A button has a rectangular shape. It can be selected and pressed.
- `ScreenshotButton extends Button`. Every ScreenshotButton object is a clickable button. It allows the user to take a screenshot of the display window of the game when it is pressed.
- `RestartGameButton extends Button`. Every RestartGameButton object is a clickable button. It allows the user to restart the game.

4 Create the TreasureHunt class

Let's create a new class called `TreasureHunt` which contains a `main` method.

- Your `TreasureHunt` class MUST inherit from a class defined within the provided processing core jar file called `PApplet` . You'll need to import processing.core.PApplet for this to work). This makes your TreasureHunt class to BE of type PApplet.
- Recall that PApplet models the Processing window for a graphic application or program. It provides good ways to create a new window and draw graphics onto it.
- In your whole graphic application, only ONE PApplet object must be used. It is an instance of this `TreasureHunt` class. This means that within all the instance methods, the keyword `this` reference is of type `TreasureHunt` which is also of type `PApplet`.
- Keep in mind that all clickable objects including interactive objects and buttons which will be created later will be drawn in **this** `TreasureHunt` PApplet object.

4.1 TreasureHunt's main() method

Now, define a main method to include the following single statement ONLY.

```
PApplet.main("TreasureHunt"); // do not add any other statement to the main method
// The PApplet.main() method takes a String input parameter which represents
// the name of your PApplet class.
```

This should result in a program that opens a small window when it is run similar to the one shown in Figure 4. We won't be adding anything more to this main method for the rest of this assignment.

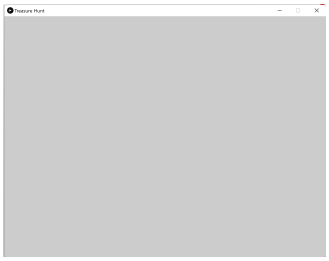


Figure 4: Small Empty Window

4.2 TreasureHunt's data fields

Your `TreasureHunt` class must contain the following **instance** data fields at the top of the class and outside of any method.

```
private PImage backgroundImage; // PImage object which represents the background image
private ArrayList<Clickable> gameObjects; // list storing objects instances of Clickable
```

- Do NOT initialize any of the above data fields where they are defined. They will be initialized in the `setup()` method.
- You can notice that none of the above data fields is static. They are ALL instance data fields.
- Notice also that these two data fields are private, meaning that they are not directly accessible from the outside of the class `TreasureHunt`.

4.3 TreasureHunt's public instance methods

Your TreasureHuntclass will implement the instance methods defined in these javadocs. Note that we are NOT going to implement all these methods at this level. We'll start by implementing

the methods defined in this file. Copy and paste the provided code within your `TreasureHunt` class below the `main` method.

- Read carefully through the provided implementation of the TreasureHunt instance methods and check how the settings of the graphic application have been made.
- The overridden `settings()` method sets the width and height dimensions of the display application of our game.
- The overridden `setup()` method initializes the graphic settings of our graphic application and calls the instance method `initGame()`.
- Notice carefully that we set the the `imageMode` and `rectMode` to `PApplet.CORNERS`. This means that images and rectangles are going to be drawn at the **x and y positions** corresponding to the location of the **upper-left corner** of the image or the rectangle within the display window.
- Also, notice that all the public methods defined in the `PApplet` class are directly accessible in its child class `TreasureHunt`.
- Complete the implementation of the provided methods with respect to the provided requirements in the `TODO` tags and the comments of the javadoc style method headers, and following the details provided below.
- Within the `initGame` method, use the PApplet.loadImage() method to load the background image and assign its reference to the `backgroundImage` instance data field.
- Consider the following string as input to the `PApplet.loadImage()` instance method while loading the background image.

```
"images" + File.separator + "background.png"
```

- Then, draw the `backgroundImage` to position (0,0) at the beginning of the draw() method by calling the PApplet.image() method.
- **DO NOT** complete the third algorithmic step defined by the third `TODO` tag within the `draw()` method related to removing deactivated interactive objects from the `gameObjects` list. We are going to complete this step after creating the `InteractiveObject` class.
- Running your program should now result in a window showing the image of the backyard or playground depicted above (without any of the fun interactive objects) as shown in Figure 5.
- Next, implement the following methods.

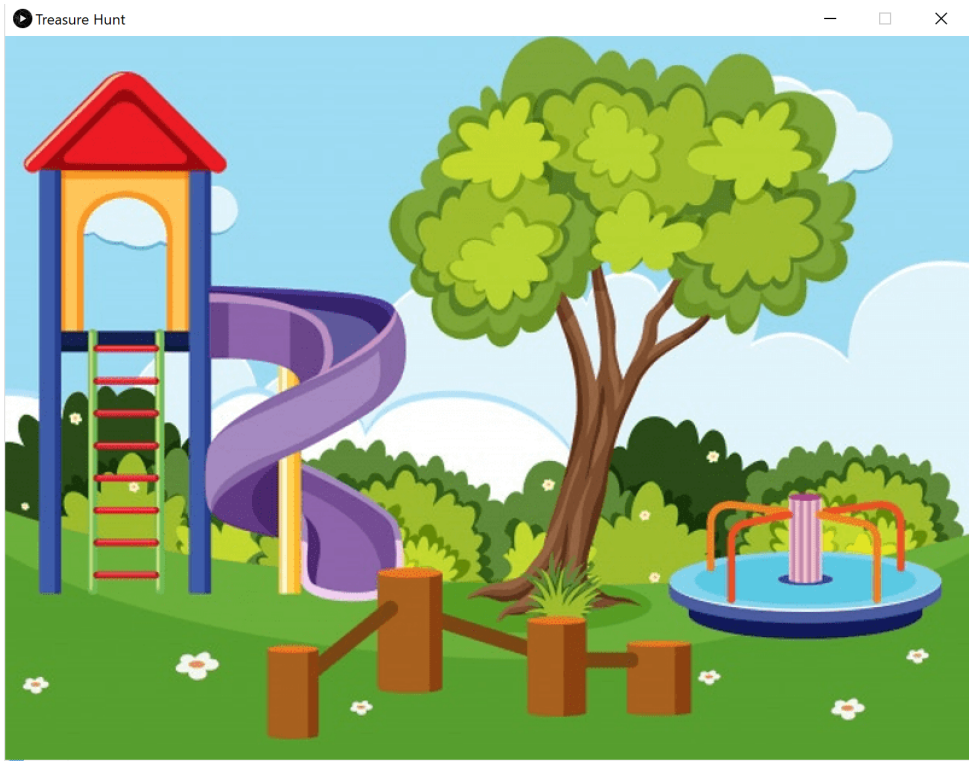


Figure 5: Treasure hunt game display window with background image only

- The `TreasureHunt.add()` method which adds the passed clickableObject input to the `gameObjects` arraylist.
- The second algorithmic step within the `TreasureHunt.draw()` method to call the `draw()` method from every reference stored within the arraylist `gameObjects`.
- The implementation details of `TreasureHunt.mousePressed()` and `mouseReleased()` methods. We will NOT need to make any changes to the implementation of these two methods later. We'll rely on the power of `polymorphism` to call the overridden behavior of these methods with respect to the most specific type of the objects whose references are stored in the `gameObjects` arraylist and upcast to the abstract type `Clickable`.
- **AVOID** using the enhanced for-each loop to traverse the arraylist of game objects and call the `mousePressed` or `mouseReleased` or `draw` methods on each object stored in the list. An exception will be thrown later if any of these method calls will make changes to the contents of the list. **Make sure to use a classic for loop instead!**
- At this step, your code must compile with no errors, even though no object is yet added to the `gameObjects` arraylist and none of the abstract methods defined in the `Clickable` interface is implemented by any class. This is amazing. That's the power of polymorphism.

5 Create the InteractiveObject class

Now, create a new class called `InteractiveObject` which implements the `Clickable` interface and does NOT contain a main method. This class represents the **base** class of all the interactive objects in our Treasure Hunt adventure style game. A clickable interactive object can be displayed (drawn) to the display window of this application and responds to the mouse clicks.

5.1 Interactive Object's data fields

Your `InteractiveObject` class must define the following data fields outside of any method.

```
// reference to the PApplet where this object will be drawn
protected static TreasureHunt processing;
private final String NAME; // name of this interactive object
protected PImage image; // reference to the image of this object
private int x; // x-position of this interactive object in the screen
private int y; // y-position of this interactive object in the screen
  // Note that the position (x,y) of the interactive object is the position
  // of the upper-left corner of its image (and NOT the center of the image).

private boolean isActive; // tells whether or not this object is active
private boolean wasClicked; // tells whether this object was already clicked
private String message; // message to be displayed when this object is clicked
private InteractiveObject nextClue; // Object to be activated when this object is
                                    // clicked the first time, if any
```

- Notice that the `processing` data field is declared to be `static`, meaning that all the interactive objects are drawn and interact on the same display window. It is also a `protected` data field, meaning that it is directly accessible to all the subclasses of the `InteractiveObject` class.
- Every interactive object has a `final` name, an image of type PImage, x and y positions, and a message to be displayed when the object is clicked.
- Some interactive objects can trigger a next clue when they are clicked the first time (another interactive object to appear in the game display window).
- The `wasClicked` data field indicates whether the interactive object has been clicked at least one time or not.
- The `isActive` data field indicates whether the interactive object is active or not.

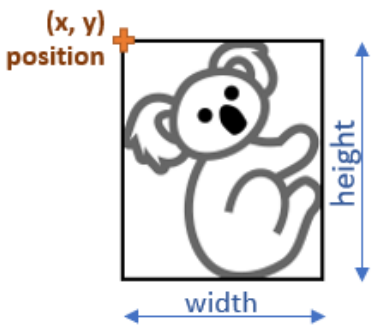
5.2 InteractiveObject's behaviors

Start by defining the above data fields. Then, implement the constructors and public methods conforming to the details provided in the javadoc of the InteractiveObject class. In these javadocs, click on each constructor, or method to go to its detailed java documentation.

- The `InteractiveObject` class MUST override all the abstract methods defined in the `Clickable` interface.
- In the constructor of the `InteractiveObject` class, use the `PApplet.loadImage()` method to load the image of the object and assign the returned reference of type `PImage` to the image instance field. The image file name to be passed to the `loadImage()` method call is the following, where `name` is the name of the interactive object.

```
"images" + File.separator + name + ".png";
```

- Notice that when created, an interactive object must be set to be active and not yet clicked.
- If the constructor is passed a next clue as input, that next clue must be deactivated by the constructor.
- Try to avoid code redundancy within the implementation of the overloaded constructors by appropriately using calling `this()` (constructor call of the same class with a specific list of input.
- After implementing the constructors of the class `InteractiveObject`, we highly recommend implementing the setter and getter methods. Next, you can implement the overridden methods. Please refer to the comments provided in the javadoc of the class InteractiveObject for details on how each method is expected to operate.
- To determine whether is mouse is over the interactive object (the `isMouseOver()` method), we recommend using the following data fields.
  - The x and y positions of the interactive object refer to the position of the **upper-left corner** of its image in the display window as illustrated in the image below.



- You can get the **width** and **height** of the interactive object using the reference to its image (`image` of type `PImage`). Every `PImage` object defines two public instance fields named `width` and `height` which represent the width and height of the image, respectively.
- The `processing.mouseX` and `processing.mouseY` represent the x and y positions of the mouse in the display window.

6 TreasureHunt: Adding and removing Interactive Objects

6.1 Adding interactive objects

Now, let's try to add an interactive object to our treasure hunt game, and check whether it is appropriately drawn, and whether its message is displayed on the console each time it is clicked.

- First, set the display window where all the Interactive Objects will be drawn and interact. Within the `TreasureHunt.setup()` method call the `InteractiveObject.setProcessing()` static method and passe it a reference to the **current** `TreasureHunt` object. Recall that the reference `this` represents a reference to the current object within a class. It can be used in the constructor of the class and within any instance method.
- Make sure to set the processing graphic display window of the `InteractiveObject` class in the `setup()` method BEFORE the method call to `initGame()`.
- Next, within the `initGame()` method and after initializing the `gameObjects` list to an empty arraylist of `Clickable`, create the following object and ADD it to the list.

```
new InteractiveObject("koala", 350, 65,
    "What a cute stuffed koala! It looks like a real one!");
```

  The above constructor call should create a new InteractiveObject named koala. The koala should be drawn at position (350, 65) which represents the x, y position of the upper-left corner of its image. The message "What a cute stuffed koala! It looks like a real one!" must be displayed to the console each time the image of the koala is clicked.
- After adding the koala to the `gameObjects` arraylist, and without making any changes to the implementation of `draw` and mousePressed defined in the `TreasureHunt` method, you should be run your program. Your program run must satisfy the expected behavior of the koala as shown in Figure 6.

Figure 6: Koala: an Interactive Object added to the game and clicked!

6.2 Finding interactive objects by name

Now, you can add and implement the `protected` helper method named `findObjectByName` conforming to the details provided in this javadoc.

6.3 Adding more interactive objects and remove deactivated ones

Now, let's try adding an Interactive object which activates a clue when clicked (another interactive object will be activated when clicked).

- First, within the `TreasureHunt.initGame()` method, create a new `InteractiveObject` called `coinTarget` with the following properties. Then, add it to the list of game objects. The `coinTarget` does not have a next clue.

```
name: "coinTarget"
x-position: 65
y-position: 510
message: "This is where the gold coin must be dropped."
```

- Now, if you run the program in addition to the koala, a coin target will appear to the screen as illustrated in Figure 7. Also, the message assigned to the coin target must be displayed on the console each time the coin target is clicked.

Figure 7: Coin target added to the game and clicked

- Next, create a new `InteractiveObject` called `map` with the following properties. Then, add it to the list of game objects. .

```
name: "map"
x-position: 330
y-position: 550
message: "The map says to drop the gold coin there."
nextClue: coinTarget
```

- Now, if you run your program, you can see the koala, the map, and the coinTarget drawn on the display window. When you click on any of these interactive objects, you should have their associated message displayed on the console. But ...!! One minute! Did you notice that something is WRONG! The `coinTarget` is supposed to appear on the display window ONLY after clicking on the `map` for the first time. But it is already there.
- First consider checking whether the constructor `InteractiveObject(String, int, int, String, InteractiveObject)` sets the `nextClue` data field to the passed nextClue input parameter and deactivates it.
- Next, within the same class `InteractiveObject`, double check that the `mousePressed` method activates `nextClue` if the `InteractiveObject` has a next clue and it was not clicked.
- We would like to keep the size of the arraylist `gameObjects` as small as possible, such that it does NOT store deactivated interactive objects.So, we should now go back to the

`TreasureHunt` class and make sure to remove deactivated objects. That is the job of the `TreasureHunt.draw()` method and more specifically the third `TODO` tag to complete in that method. After calling draw() on every interactive object in gameObjects list, the TreasureHunt's draw() method should search through and remove all deactivated objects from the array list.

- Hints on how to remove specific objects from an array list using a **classic for loop** are provided in `Question5` of the **p05 pre-assignment quiz**. Note that you MUST use a classic for-loop to traverse the `gameObjects` list to remove any non-active `InteractiveObject` stored in the list and NOT the enhanced for-each loop.
- Notice carefully that the `gameObjects` list stores references of type `Clickable`. Clickable objects include buttons which are NOT instance of `InteractiveObject`. They will be added later into the list of game objects.
- After implementing this, your program run should result into a koala and a map. When you click **for the first time** on the map, the coin target will appear on the screen.
- Clicking on any of the interactive objects will result in their associated message displayed on the console.
- Note: You may have implemented `InteractiveObject.draw()` method to draw the image of the interactive object only if it IS active. That is NOT required. But to be consistent, if you implemented such behavior, you may need to update the mousePressed overridden in the InteractiveObject class to also do nothing if the object is NOT active.

7 Draggable Objects

Now create a class called `DraggableObject` which **extends** the `InteractiveObject` class. This class models interactive objects which can be dragged in the screen.

- The `DraggableObject` class must define all the data fields defined in these javadocs with the exact name, type, and access modifiers.
- Create and implement the two constructors of this class conforming to the details provided in these javadocs.
- Create and all the methods defined in the DraggableObject class conforming to the provided specification in their javadoc style method headers.
- Recall that `processing.mouseX` and `processing.mouseY` store the current position of the mouse. Notice that the protected data field processing defined in the super class `InteractiveObject` is directly accessible.
- When the mouse is pressed and it is over the draggable object, start dragging it.

- When the mouse is released, stop dragging the object.
- To draw a draggable object (within the `draw()` method), check first whether the object is dragging. If this object is being dragged, it should be moved by the same amount that the mouse has moved between this and the previous call of the draw() method (use oldMouseX and oldMouseY to track this). After making any update to the object's position, draw the image of the object to its current position.

8 Droppable Objects

Now, create a new class called `DroppableObject` which extends `DraggableObject`, and allows us to specify a target for this kind of interactive object to be dropped on along with an action that is produced when this happens.

- The DroppableObject class defines a single private instance data field called target.

```
private InteractiveObject target; // target of this droppable object
```

- Add and implement the constructor of the class DroppableObject and its defined methods conforming to the details provided in these javadocs
- To implement the `isOver()` method which checks whether this DroppableObject overlaps with another InteractiveObject, you can refer to the following hints.
- There are several resources online which can help you implement the behavior of isOver() to check whether the image of an object overlaps with the image of another object. You can use the key search ("check rectangles for overlap"). Here is one of the online resources that you can refer to. Notice also that the position (x,y) of the images of our interactive objects is the position the **upper-left corner** of the image.

9 Load game settings from a file of clues

You now have code in place that will enable you to create a variety of different puzzles for many different kinds of adventure games. Creating these puzzles and games will require fun ideas and corresponding graphics. Rather than hard-coding the interactive objects for a specific puzzle into our TreasureHunt's initGame() method called from the `setup()`,let's load the background image filename, and introductory text message, and interactive objects or clues descriptions from a text file.

- Copy the method definitions for `loadGameSettings()` and the accompanying helper methods from this file into your `TreasureHunt` class.

- Then, call the following statement from `initGame()`, **instead of** directly loading the background image or creating any interactive objects objects from there.

```
loadGameSettings("clues" + File.separator + "treasureHunt.clues");
```

- It's not required that you play through this provided game, but it is a nice way to test some of the functionality that you have implemented.

10 Add Clickable Buttons to the Treasure Hunt application

Now for the last part of this assignment, we want to add two buttons to allow the players to either capture progress as a screenshot or restart the game. We provide you in this file which contains a starter code for a class called `Button` which implements the interface `Clickable`. Download the provided class and add it to your project.

10.1 Button class

The class `Button` represents the base class for any clickable button which can be added to a processing graphical application. Read through the provided implementation and notice the following.

- Every Button object has a final width, a final height, x and y positions, and a label.
- All buttons share the same graphic display window `processing` protected data field of type `PApplet` (and NOT of type `TreasureHunt`).
- A button is drawn as a rectangle.
- The color of a button switches from dark blue to light blue depending whether the button is selected or not.
- Same thing, the color of the text of a button changes from black to white when the button is selected and vice versa.
- You are recommended but not required to explore the following methods defined in the `PApplet` class used to draw a button.
  - rect(): This is the processing method for drawing a rectangle. It takes four parameters: 1) the x-position of the upper left corner, 2) the y-position of the upper left corner, 3) the x-position of the lower right corner, 4) the y-position of the lower right corner. These corner positions must be calculated so that the button appears centered around its position (x,y) fields with the appropriate WIDTH and HEIGHT.
  - fill(): This method is used to set the drawing color.

  - text(): This processing method is used to draw a text to the display window. The PApplet.text() method takes three parameters: 1) the string of text to draw, 2) the x-position that this text should be centered around, and 3) the y-position that this text should be centered around.

- After getting a good understanding of the provided code, complete the implementation of the `Button.isMouseOver` method conforming to the details provided in its javadoc style method header.

- Next, in the `TreasureHunt.setup` method make sure to call `Button.setProcessing()` to set the display window of all button objects to `this` TreasureHunt PApplet object, before calling the `initGame` method.

10.2 Derived Button classes

Now, create the following new classes that MUST extend the provided `Button` base class. This makes them all of type `Clickable` since the Button class implements that interface.

- `RestartGameButton` extends `Button`: Reset progress and restart the game

- `ScreenshotButton` extends `Button`: Capture a screenshot of progress in the game

The starter code of the methods to be implemented in the Button is provided in this file. Download this file and add it to your project with the existing source files. You will need to complete the `isMouseOver()` method within the Button class. After implementing the Button method, go to the `TreasureHunt.setup()` method and call the `Button.setProcessing()` method with the appropriate input argument so that all buttons will be displayed and will operate in the current `TreasureHunt` object. Both the classes above must define the following instance methods. No specific data fields are defined in these classes.

- One constructor which takes two integer variables as input parameters. The first integer represents the x-position of the button. The second integer represents the y-position of the button. Each of these constructors calls the constructor of the superclass Button (String,float,float). The String labels of the two specific buttons are as follows.

  ```
  "Take a screenshot" for the ScreenshotButton,
  "Restart Game" for the RestartGameButton.
  ```

- Only the public instance method mousePressed() must be overridden in each if the 2 specific button classes.

- Each time the RestartGameButton is pressed, the `TreasureHunt.initGame()` must be called to reset progress and return to initialization state.

- Each time the ScreenshotButton is pressed, the current state of the game should be saved in an image with the filename *screenshot.png*.

- Use the `PApplet.save(String filename)` to save a capture of the display window.

Next, in the `TreasureHunt.initGame()` method, create two buttons at the following positions, and add them to the objects arraylist.

- an RestartGameButton at position (0, 0),
- as ScreenshotButton at position (140, 0).

Now, you can run your application. You should have an interactive display window comparable to the one illustrated by the demo video at the top of this write-up.

11 Assignment Submission

Congratulations on finishing this CS300 assignment! After verifying that your work is correct, and written clearly in a style that is consistent with the CS300 Course Style Guide, you should submit your final work through Gradescope. The only EIGHT file that you must submit is `TreasureHunt.java`, `Clickable.java`, `InteractiveObject.java`, `DraggableObject.java`, `DroppableObject.java`, `Button.java`, `RestartGameButton.java`, `ScreenshotButton.java`. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of **9:59PM on March 24**th. The second portion of your grade for this assignment will be determined by running that same submission against additional offline automated grading tests after the submission deadline.